Article:

Dörflinger A., Albers M., Fiethe B., Michalik H. (2018) Hardware Acceleration in Genode OS Using Dynamic Partial Reconfiguration. In: Berekovic M., Buchty R., Hamann H., Koch D., Pionteck T. (eds) Architecture of Computing Systems – ARCS 2018. ARCS 2018. Lecture Notes in Computer Science, vol 10793. Springer, Cham.

DOI: 10.1007/978-3-319-77610-1_21

Hardware Acceleration in Genode OS Using Dynamic Partial Reconfiguration

Alexander Dörflinger, Mark Albers, Björn Fiethe, and Harald Michalik

Institute of Computer and Network Engineering (IDA)
TU Braunschweig
Braunschweig, Germany
{doerflinger, albers, fiethe, michalik}@ida.ing.tu-bs.de

Abstract. Algorithms with operations on large regular data structures such as image processing can be highly accelerated when executed as hardware tasks in an FPGA fabric. The Dynamic Partial Reconfiguration (DPR) feature of new SRAM-based FPGA families allows a dynamic swapping and replacement of hardware tasks during runtime. Particularly embedded systems with processing chains that change over time or that are too large to be implemented in an FPGA fabric in parallel, benefit from DPR. In this paper we present a complete framework for hardware acceleration using DPR in the microkernel based Genode OS. This makes the DPR feature available not only for the high-performance computing field, but also for safety-critical applications. The new framework is evaluated for an exemplary imaging application running on a Xilinx Zynq-7000 SoC.

1 Introduction

Dynamic Partial Reconfiguration (DPR) is a promising feature of new SRAM-based FPGAs to increase the overall processing power of a system. It allows to offload software tasks and process them as hardware tasks within the FPGA fabric. Computation-intensive algorithms as needed e.g. for computer vision systems yield high acceleration rates when executed in hardware [1]. Without DPR, all hardware tasks needed at some point during runtime, have to be instantiated concurrently in a static FPGA design. Due to limited resources available, only a few tasks could be migrated to hardware. DPR now allows to time-share resources of the FPGA by swapping hardware tasks in reconfigurable regions. Therefore, it combines the performance gain of hardware acceleration with the flexibility of software tasks. Furthermore, complex processing pipelines that would not fit in one static FPGA design can now be implemented for sequential execution.

Robotic applications and embedded systems in general have strict requirements regarding the utilized operating system in matters of real-time, safety and reliability. The Genode OS [2] targets safety-critical applications because it enforces a strong isolation between software components. For that reason, it has been decided to use Genode OS in various research projects. Specifically, the Controlling Concurrent Change (CCC) project [3] investigates mechanisms for

https://doi.org/10.24355/dbbs.084-202105041440-0

an automated integration of embedded systems. In this context, the stringent fault isolation and separation of concerns provided by Genode OS is used to border the effects of each sub-component on the overall system.

The DPR feature has been investigated in CCC for adapting a given platform to different operation scenarios, e.g. a car driving on a highway / in a city / parking. Hardware accelerators suitable for the current scenario are loaded into the FPGA fabric during runtime. This extends the utilization of DPR to mixed-critical systems. So far, the utilization of DPR has been limited to the high-performance computing field, and therefore safety- and reliability requirements have not been covered yet. Safety-critical applications require the reconfiguration process to be controlled from within an OS with appropriate real-time and reliability features, for which Genode OS might be suitable in future. In this paper we present, how the DPR feature can be made available for Genode OS running on a hybrid CPU-FPGA SoC device. A framework has been developed to dispatch tasks from software and execute them hardware-accelerated in the FPGA fabric of the SoC. Real-time aspects of DPR are discussed.

The rest of this paper is organized as follows: the principles of Genode OS are introduced in Sect. 2. Sect. 3 gives an overview of DPR support in other operating systems. Subsequently the hardware- (Sect. 4) and software architecture (Sect. 5) for using DPR in Genode OS are described. In Sect. 6, the newly developed framework is evaluated for an exemplary imaging application running on a Xilinx Zynq-7000 SoC.

2 Genode OS

The Genode OS framework [2] is a novel operating system approach, which is able to master complexity by applying a strict organizational structure to all software components including device drivers, system services and applications. Its continuing development takes place as a community-driven open source project.

2.1 Microkernel Based System Policy

A kernel of a modern operating system, such as the Linux kernel, manages resources, accesses the hardware, controls user processes, and more. Hence, it requires the privilege to control the whole machine. The high functional requirements and the broad range of existing hardware causes such a kernel to grow huge, by which it is impossible to fully avoid safety and security leaks that could corrupt the proper operation of the whole system. An isolation of concurrently running user applications can be provided by executing them within a dedicated address space and allowing interaction with other user applications only via mechanisms provided by the kernel. Microkernel-based systems use this technique also for device drivers, file systems, and other typical kernel-level services. Therefore, the effect of a bug-prone component is locally restricted. Furthermore, a microkernel enforces CPU time scheduling and can grant guaranteed processing time to user processes. No unprivileged system component is able to

https://doi.org/10.24355/dbbs.084-202105041440-0

violate such guarantees. Therefore, a microkernel can safely execute sensitive applications, unprivileged system services, and large untrusted applications side by side on one machine.

To make the approach of fault isolation and separation of concerns effective, all those unprivileged components must be appropriately organized. A policy must be provided by some instance because typical microkernels implement only mechanisms. This would be possible with a central policy management component controlled by a specially-privileged administrator. The complexity and manageability of a centralized policy, however, depends on the scale of the system. To overcome this problem, Genode OS extends the microkernel idea by decomposing also the system policy and imposes a strict organizational structure onto each part of the system. Processes are organized as a tree and child processes are created out of the resources of their respective parent. When creating a child process, a parent fully defines the virtual environment in which the new process gets executed. The child, in turn, can further create children from its assigned resources, thereby creating an arbitrary structured subsystem. Each parent maintains full control over the subsystems it created and defines their inter-relationship, for example by selectively permitting communication between them or by assigning physical resources. The parent-child interface is the same at each hierarchy level, which makes this organizational approach recursively applicable.

2.2 Component Communication

The basic communication between components takes place via services using Remote Procedure Call (RPC). In order to provide a service, a component needs to create an RPC object implementing the so-called root interface, which offers functions for creating and destroying sessions of the service. Then, the component has to inform its parent about it by an announce function, which takes the service name and the capability for the service's root interface as arguments. The counterpart of the service announcement is the creation of a session by a client which issues a session request to its parent. Along with the session call, the client specifies the type of the service and a number of session arguments. As a result of the session request, the client expects to obtain a capability to an RPC object that implements the session interface of the requested service.

3 Related Work

The approach of developing efficient embedded CPU-FPGA based systems with DPR has already been studied in some researches. [4] discusses the reconfiguration management on the Xilinx Zynq-7000 platform at application level without the use of any operating system. Another approach used a custom ARM-specified microkernel on a partial reconfigurable FPGA platform to dynamically manage reconfigurable HW accelerators and SW tasks by developing a specific scheduling mechanism [5]. Based on this, the ability to dispatch hardware tasks to virtual

https://doi.org/10.24355/dbbs.084-202105041440-0

machines hosted by the microkernel was integrated [6]. In [7], a PowerPC was used to exchange reconfigurable engines representing different image processing algorithms for driving assistant systems. There are also approaches to design new interface structures to either increase the performance [8] or reduce the resource requirements [9]. Another work describes a dynamic and partial reconfigurable system using a Zynq-7000 SoC with Linux and demonstrates, that acceptable delays for the configuration process can be achieved in that constellation [10]. In spite of all investigation efforts on DPR for CPU-FPGA based systems, there is no solution for doing this in Genode OS, yet. By implementing DPR for Genode OS, hardware acceleration can be applied for safety-critical applications which require strong isolation of software components.

4 Reconfigurable Hardware

Hardware tasks are hosted and run within separate reconfigurable regions of the FPGA fabric. These regions need to be embedded in a static logic that provides the infrastructure for communication etc. While the static logic is configured at startup and remains unchanged thereafter, the configuration of hardware tasks can be written into reconfigurable regions through the Processor Configuration Access Port (PCAP) during runtime. Other alternative configuration ports (such as ICAP, SelectMAP, Serial, and JTAG) are available in the Zynq-7000 SoC, but have drawbacks regarding bandwidth or accessibility. With a bandwidth of up to 3.2Gb/s, the PCAP allows fast reconfiguration times.

The resulting architecture for a hardware accelerated CPU-FPGA SoC design is depicted in Fig. 1. The architecture targets Xilinx Zynq-7000 or Zynq-Ultrascale+ devices with a Processing System (PS) and Programmable Logic (PL). Hardware accelerated algorithms are implemented for the FPGA fabric and can be placed in one or more available reconfigurable regions. Configuration- and status data is communicated over the AXI Lite Interconnect and attached to a general purpose AXI port (AXI GP).

Each reconfigurable region connects to an AXI Stream Interconnect network which allows flexible streaming of data to any endpoint. This allows to stream the output of one reconfigurable region directly to the input of another region, and datapaths with multiple hardware tasks to be executed sequentially can be set up. DMA (or Video DMA for image processing applications) IP-cores translate between the streaming- and memory mapped communication. For a fast transfer of processing data, a high performance AXI port (AXI HP) is used between the PS and PL.

As different hardware tasks generally have very diverse demands of FPGA resources, the definition of appropriate reconfigurable region sizes is a sophisticated problem. In order to distribute the FPGA resources between all reconfigurable regions for a given set of hardware tasks efficiently, we use an algorithm introduced by us in [11].

https://doi.org/10.24355/dbbs.084-202105041440-0

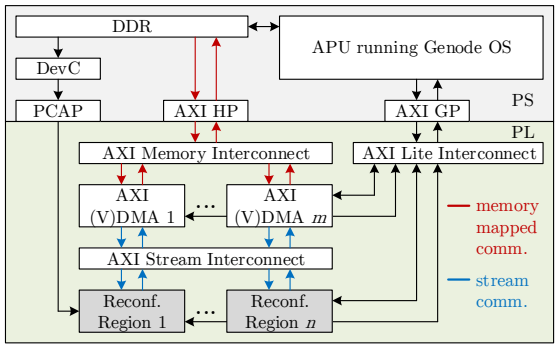

Fig. 1. Hardware accelerated CPU-FPGA SoC design.

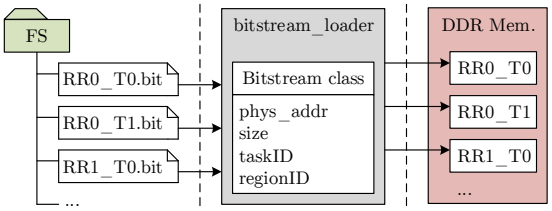

Fig. 2. Loading bitstreams from a File System to DDR memory.

5 Reconfiguration Software

5.1 Loading Partial Bitstreams

For an autonomous operation of an embedded system, all partial bitstreams need to be stored in non-volatile memory. Similar to [12], all bitstreams are copied from non-volatile memory to DDR memory during the boot sequence. Hence, partial bitstreams can be accessed rapidly and reconfiguration times are kept short. This task is executed by the *bitstream_loader* component in Genode OS. For each partial bitstream, it requests a read-only dataspace (ROM session) which is served by a file system containing the corresponding *.bit* files. Once the ROM session is created, Genode OS maps the bitstream to the private memory of the *bitstream_loader* component. The *bitstream_loader* also keeps track of metadata for each bitstream, such as its physical address, size, region, and contained hardware task. Fig. 2 depicts the principle process of loading bitstreams to DDR memory.

5.2 Accessing the Configuration Port

Once a reconfiguration process is triggered, the partial bitstream needs to be written into the FPGA fabric. This is handled by the *device configuration interface* (DevC, [13]), which moves the bitstream data from DDR memory to

https://doi.org/10.24355/dbbs.084-202105041440-0

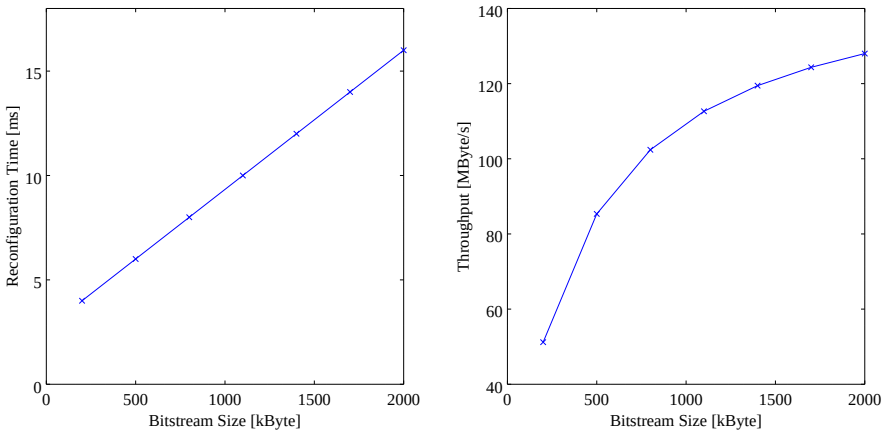

Fig. 3. Reconfiguration times and throughput for different bitstream sizes.

the PCAP using DMA. A DevC driver has been developed for Genode OS. It provides address and size of the bitstream to the DevC component, enables the PCAP port, and handles the PCAP interrupt which indicates the completion of a bitstream transfer.

Important performance metrics of the DevC driver are reconfiguration times, which are measured for various bitstream sizes ranging from 100kByte to 2MByte. As depicted in Fig. 3, the reconfiguration time scales accurately with the bitstream size. The slope of reconfiguration time to bitstream size is limited by DDR memory and the PCAP bandwidth. The initialization time of the DevC component for each reconfiguration process is independent of bitstream size and causes the throughput to drop for small bitstreams copied to the PL. The throughput saturates at about 130MByte/s for large bitstreams, which is below the PCAP bandwidth. Still, when compared to a Linux-based reference design provided by Xilinx [14] using the same FPGA device and identical PCAP clock, the reconfiguration speed achieved with the new Genode OS DevC driver is about twice as fast for a bitstream of a medium size of 734kByte.

5.3 Hardware Scheduler

The *hardware_scheduler* knows which hardware task can be placed in which reconfigurable region. Once it receives an incoming request for a hardware task, it checks if the task is already configured in any reconfigurable region and suspended. If true, it returns access to the corresponding region to the requesting software component. If the task is not configured in any region yet, it tries to place it in a free region. For Genode OS, such a *hardware_scheduler* has been developed. Up to now, it serves incoming requests in FCFS order. In future it is planned to implement an intelligent scheduling strategy and algorithm. Depending on well predictable reconfiguration times, execution times, and deadlines,

https://doi.org/10.24355/dbbs.084-202105041440-0

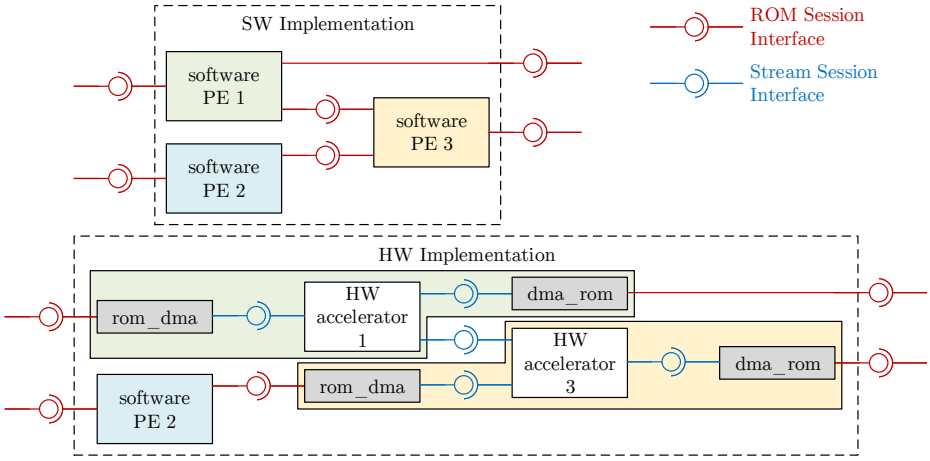

Fig. 4. SW component instantiation graphs for a processing chain fully implemented in SW and HW accelerated.

the *hardware_scheduler* can select a requested hardware task which fits best for the next reconfiguration.

The reconfiguration time of a task can be accurately predicted as it is a function of bitstream size. No other dependencies such as interfering memory access have been observed. Hence, the reconfiguration time can be upper bounded when considered in a real-time application. However, a task might get blocked once its region is occupied. In order to meet given real-time requirements, the scheduler is responsible for guaranteeing an upper bound for blocking times. The currently implemented FCFS scheduler does not satisfy this requirement and further work needs to be done on this topic.

5.4 Hardware Acceleration

The top part of Fig. 4 depicts the software component instantiation graph of a processing chain with three Processing Elements (PEs) implemented in software. Such a processing chain could be part of an image processing application. Each PE may require $1..n$ data inputs and generate $1..m$ data outputs. Multiple tasks can be connected in parallel or serially. In Genode OS, a software component can access its source data by reading from one ore more read-only dataspace(s) or ROM session(s). Respectively, the software component writes the already processed destination data to one or more RAM dataspace(s), which can be accessed by the following component as a ROM session again.

Now some computation intensive PEs are identified and should be accelerated in hardware. In the example given, PE 1 and PE 3 can be accelerated and the resulting software component instantiation graph is given in the bottom part of Fig. 4. Hardware tasks receive their source data from a network on chip and also transmit their results over the same communication medium. Therefore

https://doi.org/10.24355/dbbs.084-202105041440-0

each input dataspace needs to be converted into a stream before passing it to the hardware task. This is handled by the *rom_dma* component. It initializes a DMA engine that moves the input data from DDR memory to a network on chip, e.g. the AXI stream. The *dma_rom* component works analogously and copies stream data to a dataspace in DDR memory.

The *rom_dma* and *dma_rom* software components have been developed together with Genode OS drivers for the Xilinx AXI DMA IP-core [15]. Also, its variants *rom_vdma* and *vdma_rom* using the Xilinx AXI Video DMA IP-Core [16] are available for imaging applications.

The execution of PEs is triggered every time the data of an input ROM session gets updated. The software PE receives a notification of this event and starts processing the input data. Once it is done, it signals a notification to the proceeding PE.

For hardware accelerated designs, this forward signaling needs to be extended, because hardware modules need to be reconfigured and initialized before starting execution. The following signaling policy, as depicted in Fig. 5, is implemented:

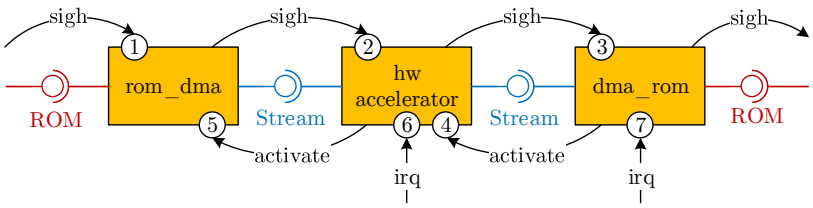

Fig. 5. Signaling policy.

1. The first component in line (here *rom_dma*) receives a notification that its input ROM has been updated. It forwards the notification to the next component in line.
2. All intermediate components in line (here *hw_accelerator*) forward the notification.
3. The last component in line (here *dma_rom*) initializes itself and signals *activate* to the preceding component. By doing this, it informs the preceding component that it is ready to stream data.
4. All intermediate components in line initialize themselves, which includes reconfiguration of hardware modules, and forward the *activate* notification. They are now ready to process data.
5. The *rom_dma* component initializes itself and starts streaming data by executing a DMA transaction.
6. On a hardware interrupt indicating the end of data processing, all hardware accelerators release their reconfigurable region.
7. On the same hardware interrupt, the next component is notified that newly processed data is available.

https://doi.org/10.24355/dbbs.084-202105041440-0

6 Exemplary Use Case and Evaluation

In order to verify the proper functionality of the new hardware acceleration framework for Genode OS and to produce realistic benchmark results, we set up the following exemplary use case. A stereo-vision system offers two operation modes in order to satisfy the requirements of different applications such as object recognition. Firstly, a high frame rate mode can be selected that is capable of providing images with a frame rate of up to 60fps. Secondly, the high-quality mode runs additionally a rectification algorithm on each image. However, in this mode only a frame rate of 30fps can be achieved. In both operation modes, a debayering algorithm converts the camera sensor data to RGB format. Both the debayering- and the rectification algorithm have been implemented as IP-cores for hardware acceleration.

Two reconfigurable regions are available for hosting hardware acceleration modules. The high frame rate requires a debayering IP-core to be placed in each region, so that sensor data from the left- and right camera eye may be processed in parallel. This operation mode emphasizes the advantage of hardware acceleration, as two debayering tasks can be executed in parallel, while an execution in software would force the tasks to be scheduled one after the other. The high-quality mode configures one debayering and one rectification IP-core; sensor data from the left- and right camera eye are processed sequentially in each hardware component. Fig. 6 depicts the hardware task graph for both operation modes.

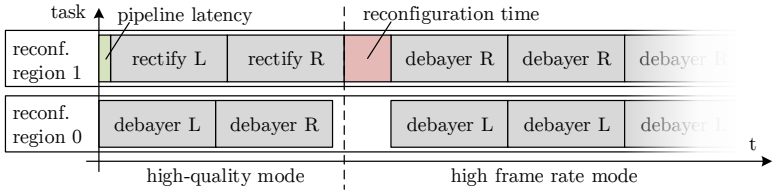

Fig. 6. Hardware task graph for high-quality and high frame rate operation modes.

For comparison, both algorithms are executed in software using the OpenCV implementation and run on one ARM Cortex-A9 core of the Zynq-7000 clocked with 667MHz. Execution times for the tasks accelerated in hardware are measured for a 100MHz clocking of the hardware modules in the FPGA fabric. Table 1 shows the results for the software- and hardware implementation of both algorithms executed stand-alone and serially one after the other. As the debayering- and rectification algorithms have no input-dependent branches, the measured execution times are very deterministic with a scatter of less than 0.2ms. All tests process images with a resolution of 1280x960 pixels.

The hardware acceleration of the debayering algorithm yields a speedup factor of about 2; the more complex rectification algorithm of about 14. When executed one after the other, the performance gain is increased even more. In

https://doi.org/10.24355/dbbs.084-202105041440-0

Table 1. Execution times

Task	SW Impl.	HW Impl.	Bitstream Size	Reconf. time
debayer	28ms	14ms	521kByte	6ms
rectify	212ms	15ms	1992kByte	16ms
debayer+rectify	241ms	15ms	2513kByte	22ms

software, both algorithms are executed sequentially, hence the execution times of debayering and rectification add up. In hardware, a stream of pixels is processed in a pipelined manner and therefore the overall execution time does not increase compared to a stand-alone execution of the debayering- or rectification algorithm.

The cost for switching the operation mode correlates to the reconfiguration time of one debayering or rectification module. As the debayering module has a quite small footprint in the FPGA fabric and therefore fits into a reconfigurable region with smaller bitstream size, its reconfiguration time is shorter. In this presented use case, exchanging a hardware module requires a reconfiguration time on a scale of its hardware execution time. To be still efficient, the number of reconfigurations needs to be minimized by a smart scheduling algorithm. The given use case drops one image when switching between one operation mode and the other, however only a few occurrences of these operation mode change are expected.

7 Conclusion

In this paper, we presented the implementation of hardware acceleration using DPR in Genode OS. For an exemplary imaging algorithm, the performance of the new framework has been evaluated. It has been showed, that the hardware acceleration yields high speedup factors while reconfiguration times are kept low. The new availability of DPR for Genode OS allows this feature to be used in safety-critical applications with strict requirements for real-time and isolation. Computation intensive algorithms implemented in Genode OS, such as in the Controlling Concurrent Change project, can now easily be accelerated in hardware.

So far, the functionality of the DPR feature in Genode OS has been verified using simple application examples. In future, we will add an intelligent hardware task scheduling for arbitrary complex applications and further performance increase.

Acknowledgment

This work is part of the DFG Research Group FOR 1800 “Controlling Concurrent Change”. Funding for the Institute of Computer and Network Engineering (IDA) was provided under grant number MI 1172/3-1.

https://doi.org/10.24355/dbbs.084-202105041440-0

References

[1] Lomuscio, A., Cardarilli, G.C., Nannarelli, A., Re, M.: A hardware framework for on-chip FPGA acceleration. In: 2016 International Symposium on Integrated Circuits (ISIC). (Dec 2016) 1–4

[2] Genode Labs: Genode OS framework. http://genode.org/ Accessed: 2017-10-12.

[3] TUBS.digital: Controlling concurrent change. http://ccc-project.org/ Accessed: 2017-10-26.

[4] Vipin, K., Fahmy, S.A.: A high speed open source controller for FPGA partial reconfiguration. In: 2012 International Conference on Field-Programmable Technology. (Dec 2012) 61–66

[5] Xia, T., Prévotet, J.C., Nouvel, F.: Microkernel dedicated for dynamic partial reconfiguration on ARM-FPGA platform. ACM SIGBED Review **11**(4) (January 2015) 31–36

[6] Xia, T., Prévotet, J.C., Nouvel, F.: Mini-nova: A lightweight ARM-based virtualization microkernel supporting dynamic partial reconfiguration. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshops. (May 2015) 71–80

[7] Claus, C., Stechele, W., Herkersdorf, A.: Autovision – a run-time reconfigurable MPSoC architecture for future driver assistance systems. **49** (June 2007) 181–187

[8] Claus, C., Zhang, B., Stechele, W., Braun, L., Hubner, M., Becker, J.: A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. In: 2008 International Conference on Field Programmable Logic and Applications. (Sept 2008) 535–538

[9] Hübner, M., Göhringer, D., Noguera, J., Becker, J.: Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs. In: IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW). (April 2010) 1–8

[10] Kadi, M.A., Rudolph, P., Göhringer, D., Hübner, M.: Dynamic and partial reconfiguration of Zynq 7000 under Linux. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig). (Dec 2013) 1–5

[11] Dörflinger, A., Fiethe, B., Michalik, H., Fekete, S.P., Keldenich, P., Scheffer, C.: Resource-efficient dynamic partial reconfiguration on FPGAs for space instruments. In: 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS). (July 2017) 24–31

[12] Xilinx, Inc.: Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices, XAPP1159. v1.0 edn. (2013)

[13] Xilinx, Inc.: Zynq-7000 All Programmable SoC TRM, UG585. v1.11 edn. (2016)

[14] Kohn, C.: Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor. Xilinx, Inc. v1.1 edn. (2015)

[15] Xilinx, Inc.: AXI DMA LogiCORE IP Product Guide, PG021. v7.1 edn. (2017)

[16] Xilinx, Inc.: AXI Video Direct Memory Access LogiCORE IP Product Guide, PG020. v6.2 edn. (2016)

https://doi.org/10.24355/dbbs.084-202105041440-0